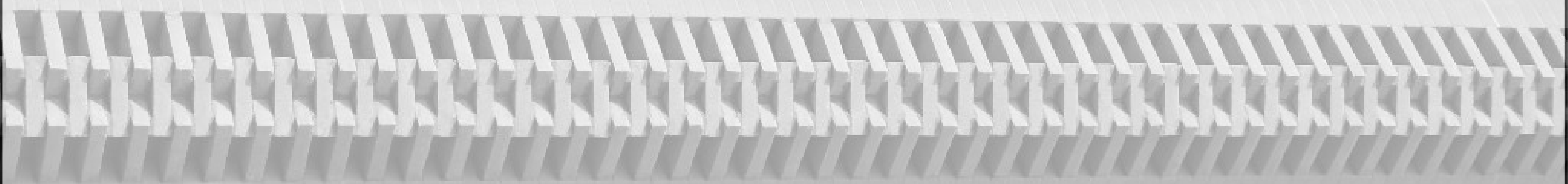
Strengthening the Zipper

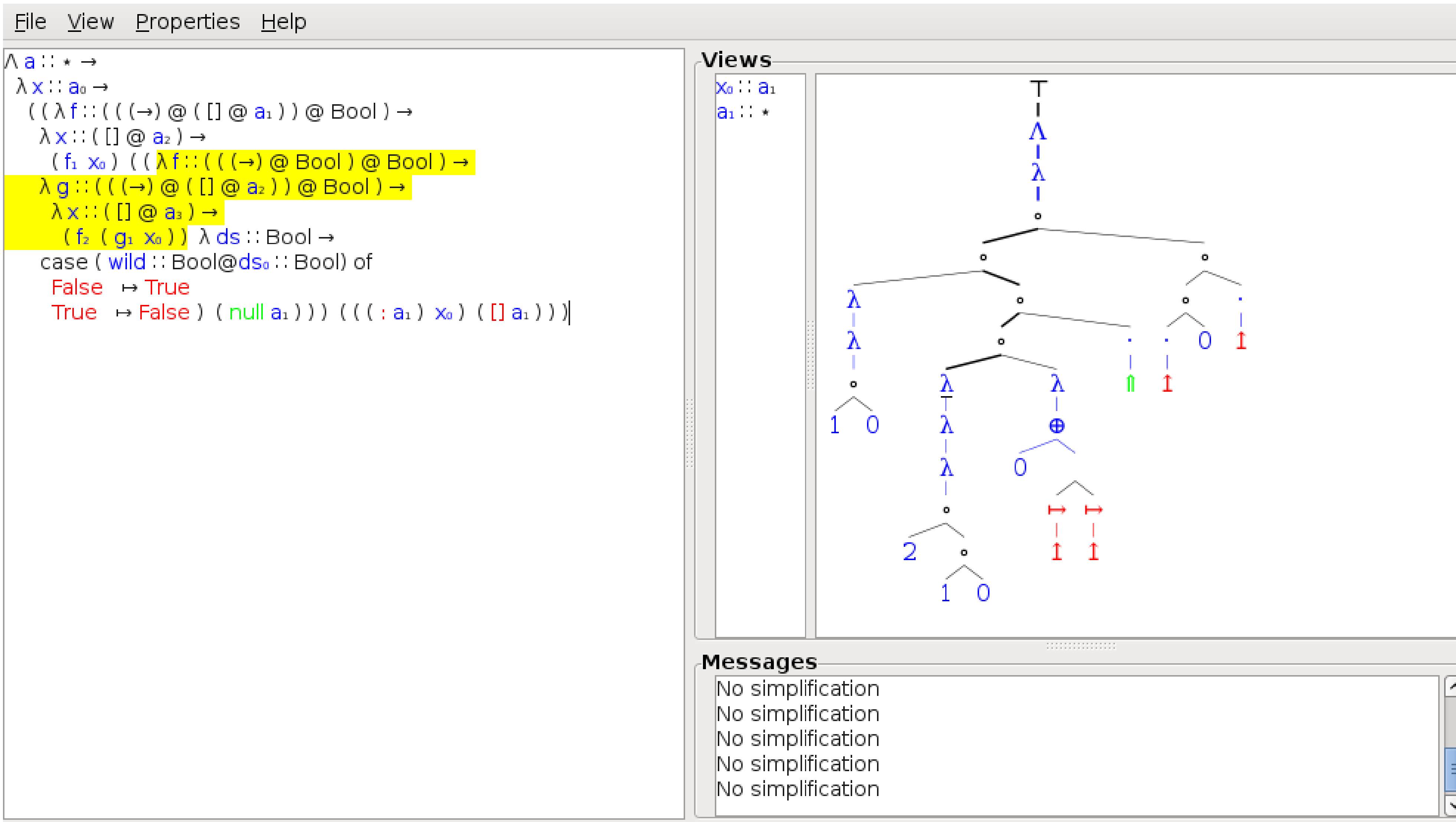
CLASE - Cursor Library for A Structured Editor



Tristan Allwood (tora@doc.ic.ac.uk)
Susan Eisenbach (s.eisenbach@imperial.ac.uk)

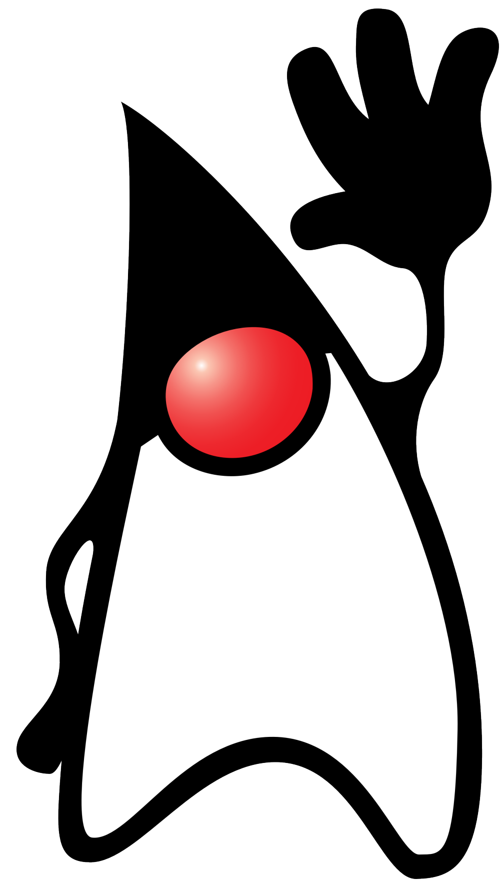
Zip! Photo from http://www.flickr.com/photos/sarmax/109561164/

Motivation


File View Properties Help
Λ a :: ⋆ →
λ x :: a_0 →
((λ f :: (((→) @ ([] @ a_1)) @ Bool) →
λ x :: ([] @ a_2) →
(f_1 x_0) ((λ f :: (((→) @ Bool) @ Bool) →
λ g :: (((→) @ ([] @ a_2)) @ Bool) →
λ x :: ([] @ a_3) →
(f_2 (g_1 x_0)) λ ds :: Bool →
case (wild :: Bool@ds_0 :: Bool) of
False ↦ True
True ↦ False) (null a_1))) (((: a_1) x_0) ([] a_1)))
Views
x_0 :: a_1
a_1 :: ⋆
Messages
No simplification
No simplification
No simplification
No simplification
No simplification

Motivation

- In place updates
- Side effects
- Pointers
- IO



Motivation

- In place updates
- Side effects
- Pointers
- IO

Towards Clase Zippers

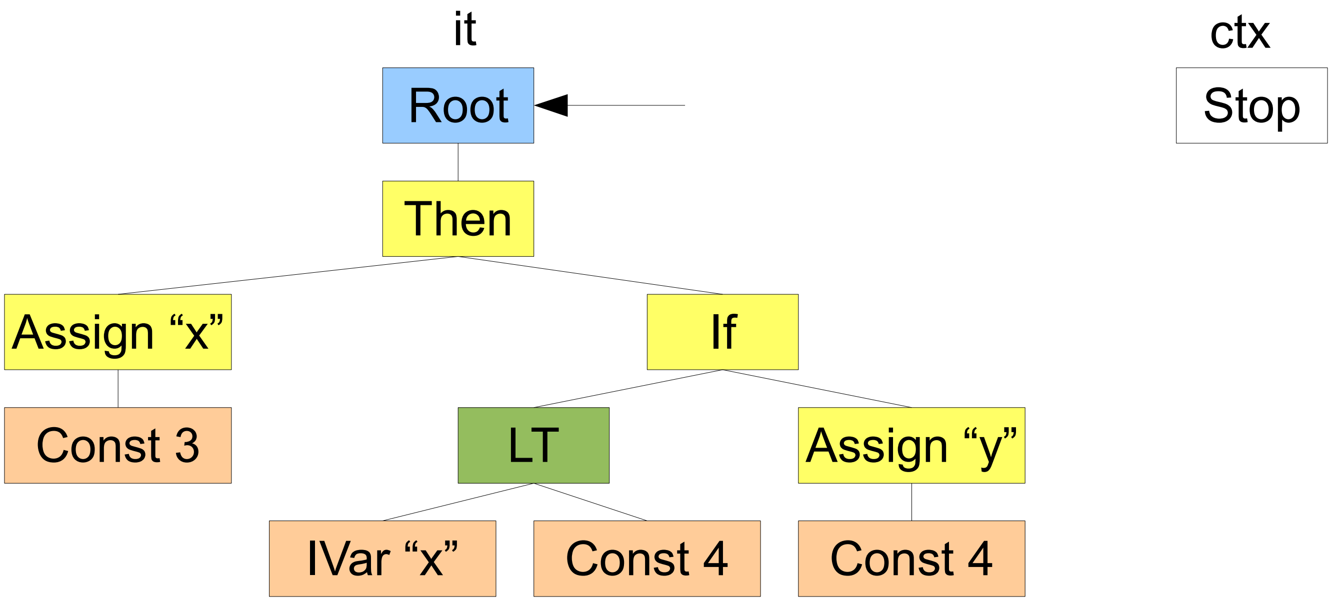
```
data Root = Root Stat
```

```
data Stat
  = Assign Var IExp
  | If BExp Stat
  | Stat `Then` Stat
```

```
data BExp
  = LT IExp IExp
  | GT IExp IExp
```

```
data IExp
  = Add IExp IExp
  | IVar Var
  | Const Int
```

Towards CLASE Zippers




```
data Root = Root Stat
```

```
data Stat
   = Assign Var IExp
   | If BExp Stat
   | Stat `Then` Stat
```

```
data BExp
   = LT IExp IExp
   | GT IExp IExp
```

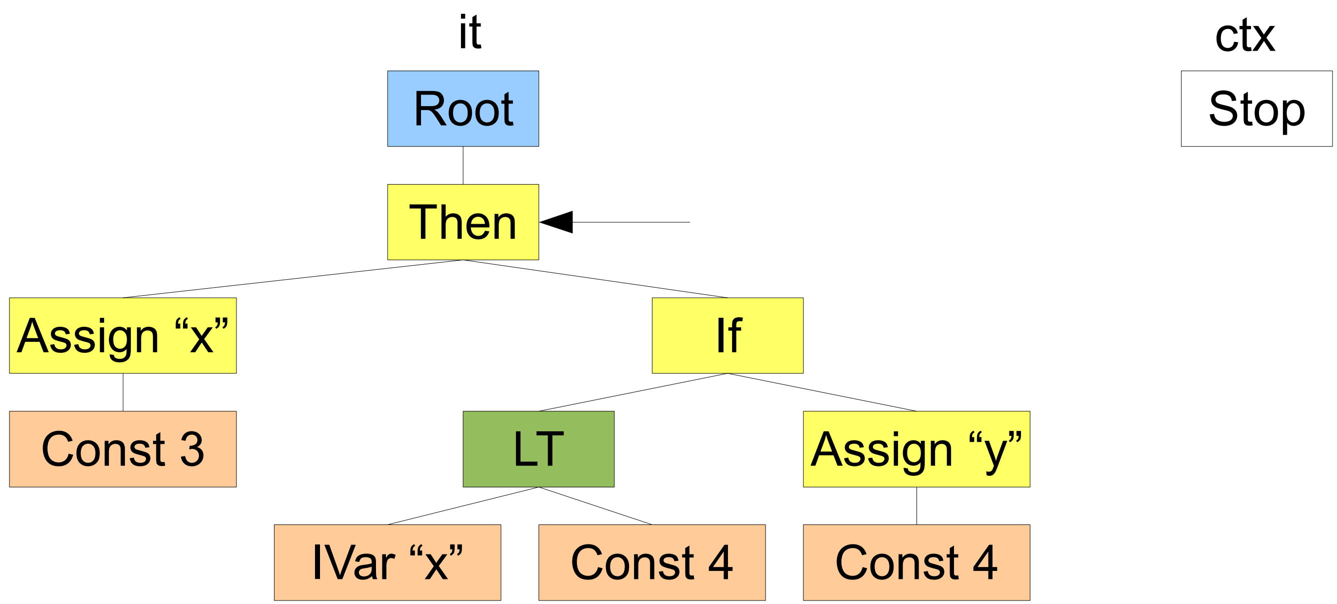
```
data IExp
   = Add IExp IExp
   | IVar Var
   | Const Int
```

```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

```
Cursor {
  it = Root ((Assign “x” (Const 3) `Then`
              (If (LT (IVar “x”) (Const 4)) (Assi
  ctx = Stop
}  :: Cursor Root
```

```
data Cursor a = Cursor {
  it :: a,
  ctx :: Path ContextI a Root
}
```

Towards CLASE Zippers




```
data Root = Root Stat
```

```
data Stat
   = Assign Var IExp
   | If BExp Stat
   | Stat `Then` Stat
```

```
data BExp
   = LT IExp IExp
   | GT IExp IExp
```

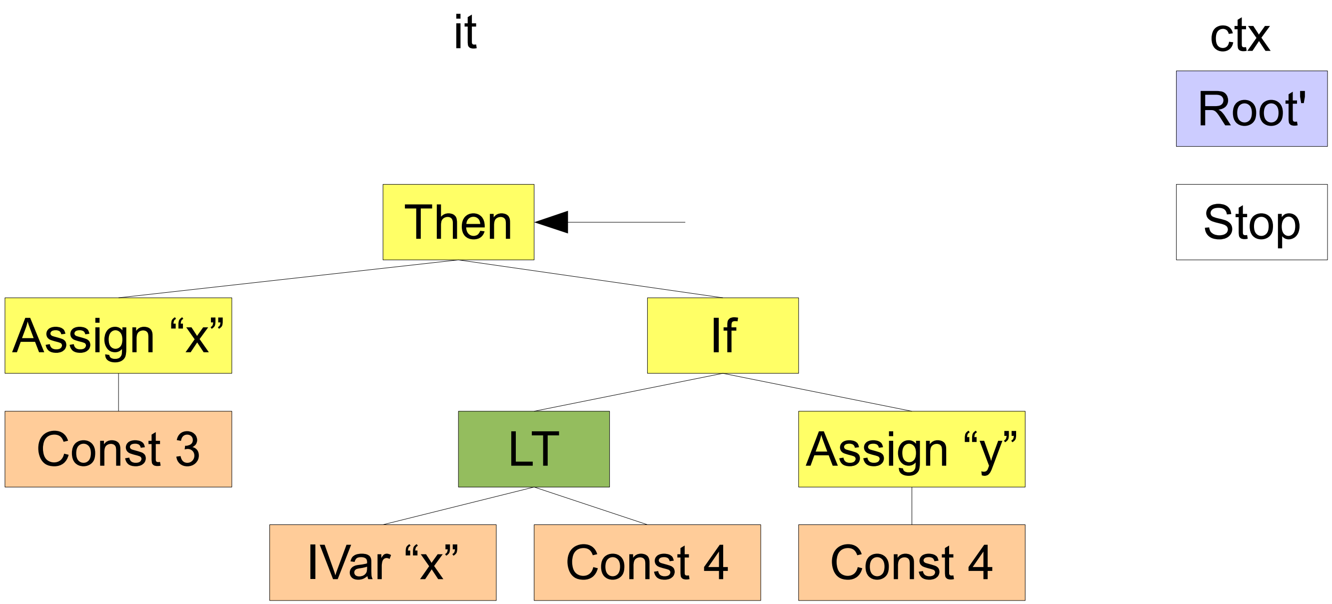
```
data IExp
   = Add IExp IExp
   | IVar Var
   | Const Int
```

```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

```
Cursor {
  it = Root ((Assign “x” (Const 3) `Then`
             (If (LT (IVar “x”) (Const 4)) (Assi
  ctx = Stop
}  :: Cursor Root
```

```
data Cursor a = Cursor {
  it :: a,
  ctx :: Path ContextI a Root
}
```

Towards CLASE Zippers




```
data Root = Root Stat
```

```
data Stat
  = Assign Var IExp
  | If BExp Stat
  | Stat `Then` Stat
```

```
data BExp
  = LT IExp IExp
  | GT IExp IExp
```

```
data IExp
  = Add IExp IExp
  | IVar Var
  | Const Int
```

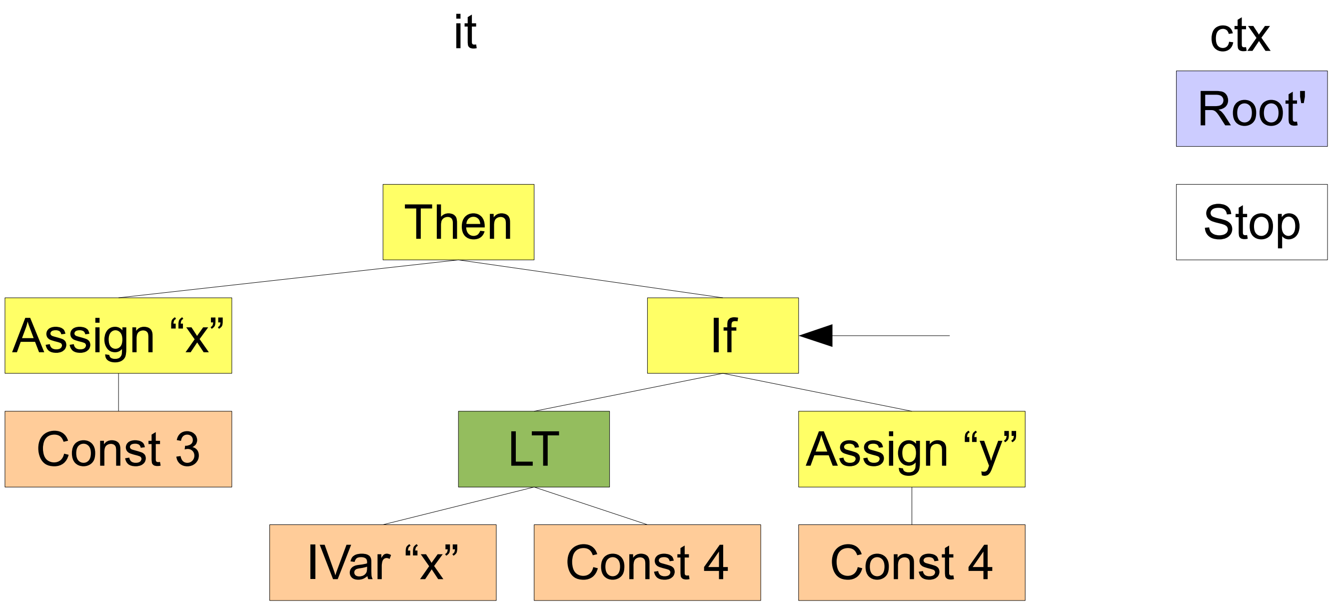
```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

```
Root' :: ContextI Stat Root
```

```
Cursor {
  it = ((Assign “x” (Const 3) `Then`
           (If (LT (IVar “x”) (Const 4)) (Assi
  ctx = Step Root' Stop
}  :: Cursor Stat
```

```
data Cursor a = Cursor {
  it :: a,
  ctx :: Path ContextI a Root
}
```

Towards CLASE Zippers




```
data Root = Root Stat
```

```
data Stat
   = Assign Var IExp
   | If BExp Stat
   | Stat `Then` Stat
```

```
data BExp
   = LT IExp IExp
   | GT IExp IExp
```

```
data IExp
   = Add IExp IExp
   | IVar Var
   | Const Int
```

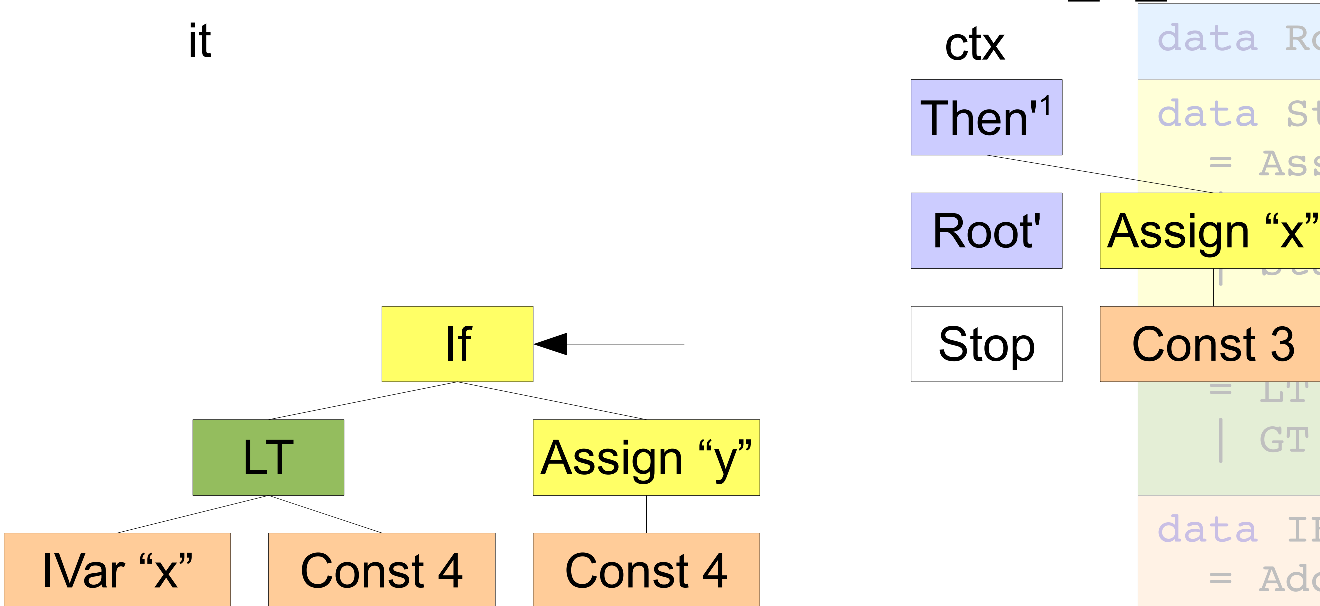
```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

```
Root' :: ContextI Stat Root
```

```
Cursor {
  it = ((Assign “x” (Const 3) `Then`
             (If (LT (IVar “x”) (Const 4)) (Assi
  ctx = Step Root' Stop
}  :: Cursor Stat
```

```
data Cursor a = Cursor {
  it :: a,
  ctx :: Path ContextI a Root
}
```

Towards CLASE Zippers




```
data Root = Root Stat

data Stat
  = Assign Var IExp
    BExp Stat
  |    `Then` Stat

       Exp
  = LT IExp IExp
  | GT IExp IExp

data IExp
  = Add IExp IExp
  | IVar Var
  | Const Int
```

```
Then'1 :: ContextI Stat Stat
Root'  :: ContextI Stat Root
```

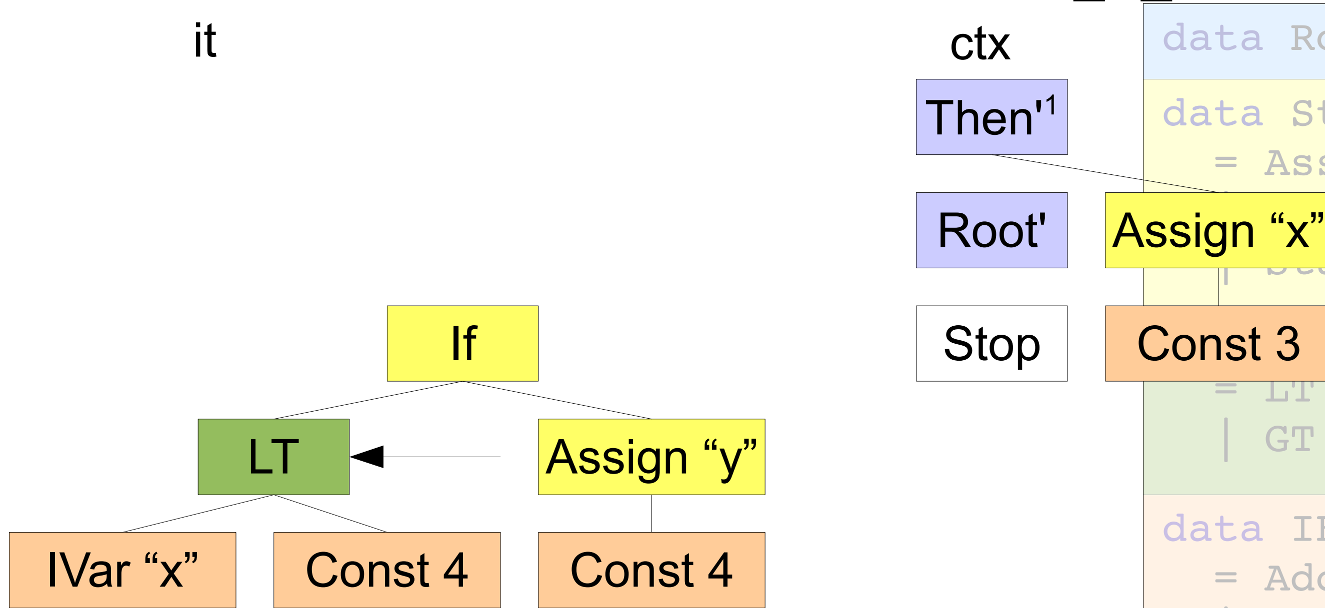
```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

```
Cursor {
  it =  If (LT (IVar “x”) (Const 4)) (Assign “y”

  ctx = Step (Then'1 (Assign “x” (Const 3))) (St
}  :: Cursor Stat
```

```
data Cursor a = Cursor {
  it :: a,
  ctx :: Path ContextI a Root
}
```

Towards CLASE Zippers




```
data Root = Root Stat

data Stat
  = Assign Var IExp
        BExp Stat
  | Stat `Then` Stat

        Exp
  = LT IExp IExp
  | GT IExp IExp

data IExp
  = Add IExp IExp
  | IVar Var
  | Const Int
```

```
Then'¹ :: ContextI Stat Stat
Root'  :: ContextI Stat Root
```

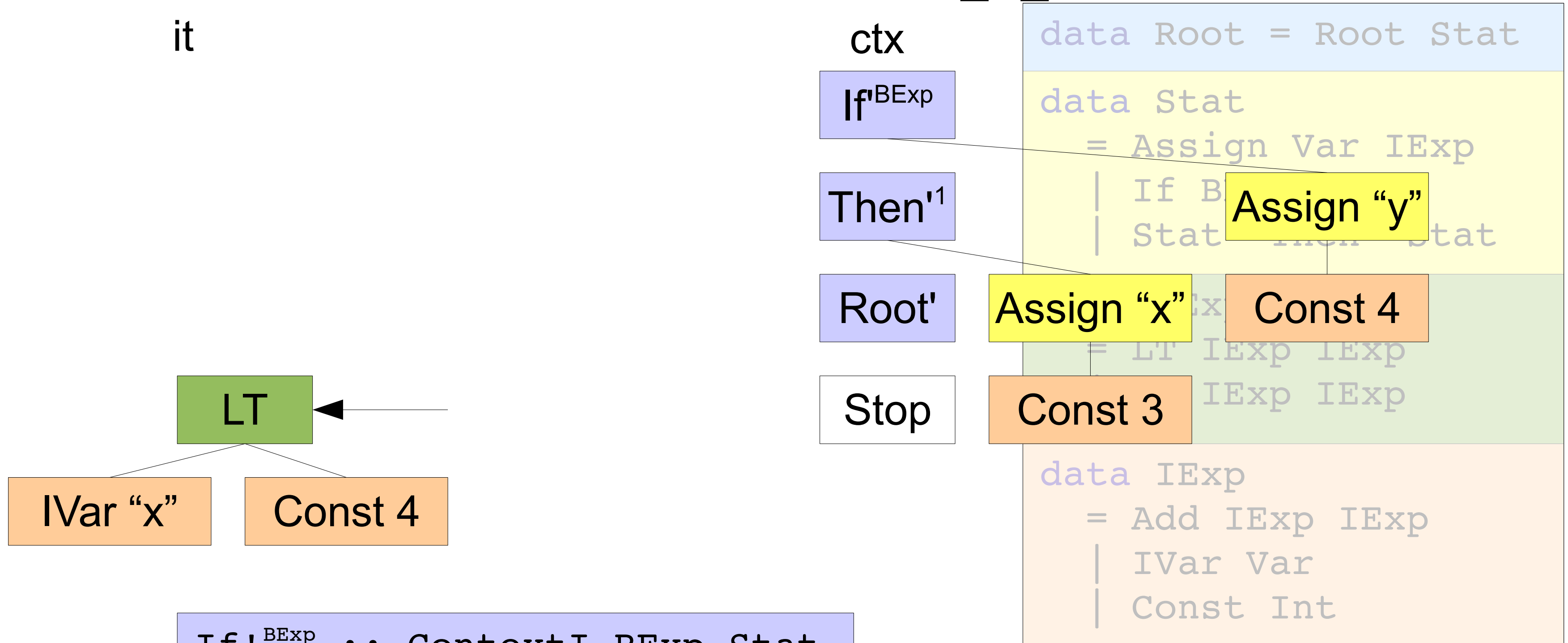
```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

```
Cursor {
  it =  If (LT (IVar “x”) (Const 4)) (Assign “y”

  ctx = Step (Then'1 (Assign “x” (Const 3))) (St
}  :: Cursor Stat
```

```
data Cursor a = Cursor {
  it :: a,
  ctx :: Path ContextI a Root
}
```

Towards CLASE Zippers



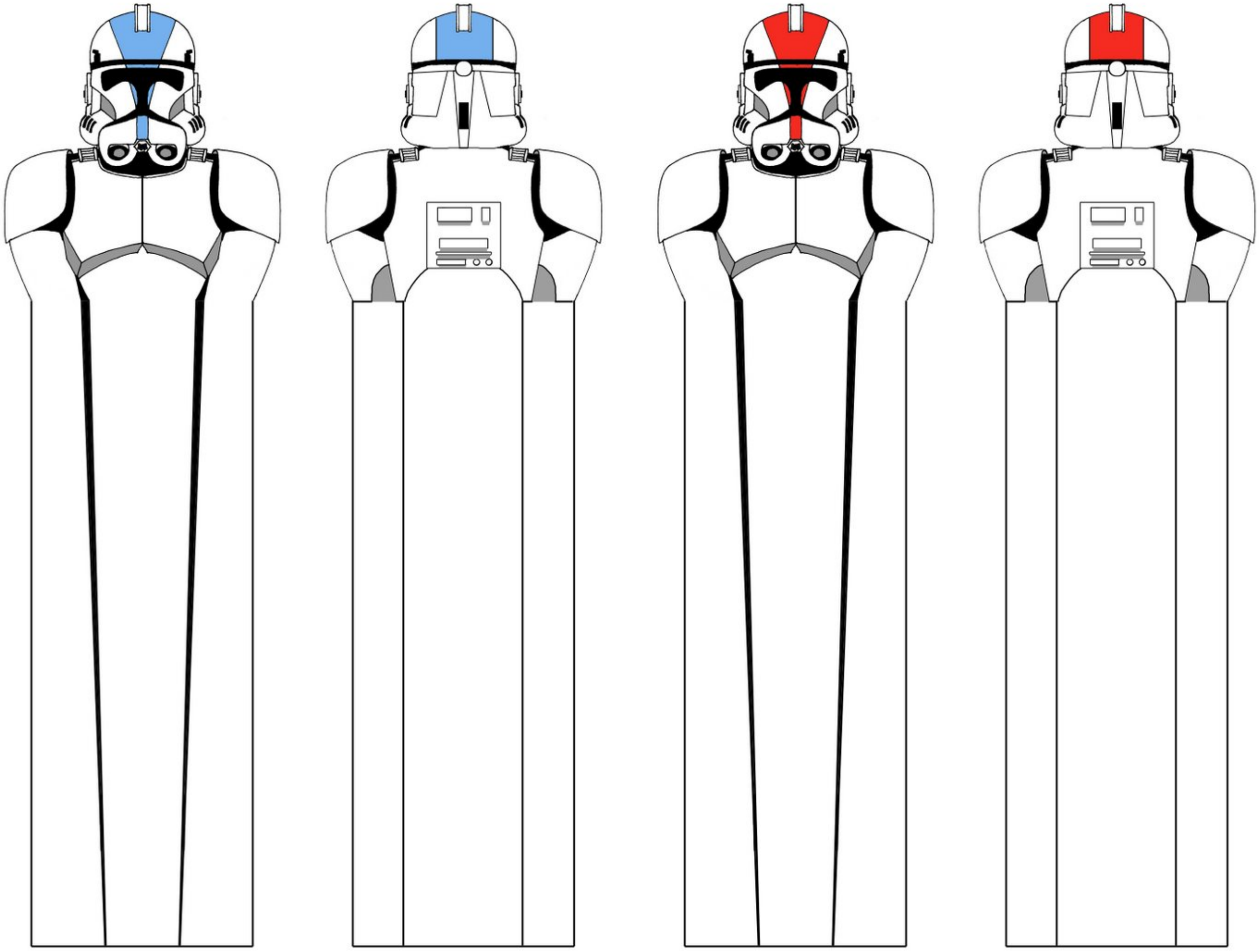

```
If'^BExp :: ContextI BExp Stat
Then'^1  :: ContextI Stat Stat
Root'    :: ContextI Stat Root
```

```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

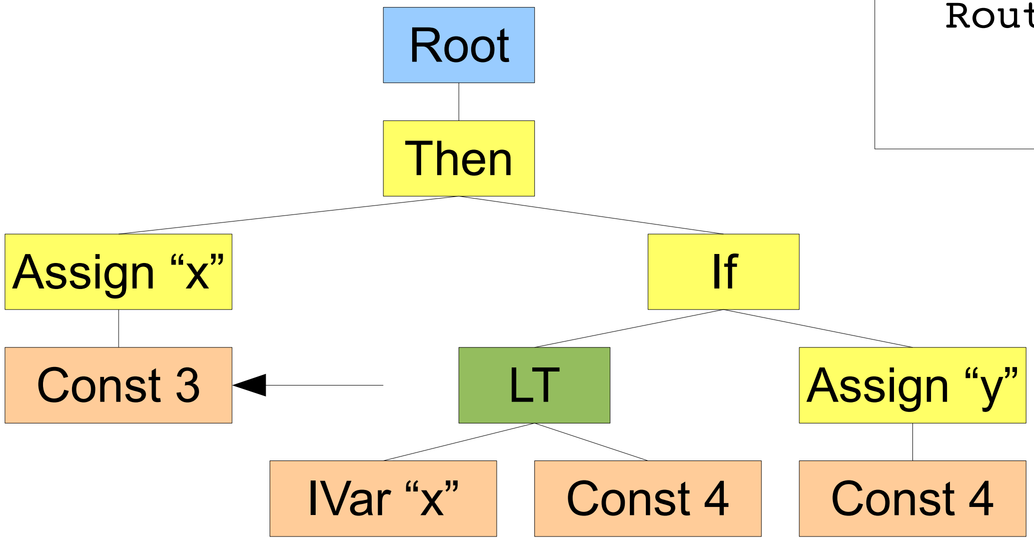
```
Cursor {
  it =  LT (IVar “x”) (Const 4),
  ctx = Step (If'^BExp (Assign “y” (Const 4))) (Ste
“x” (Const 3))) (Step Root' Stop))
}  :: Cursor BExp
```

```
data Cursor a = Cursor {
  it :: a,
  ctx :: Path ContextI a Root
}
```

Bookmarks



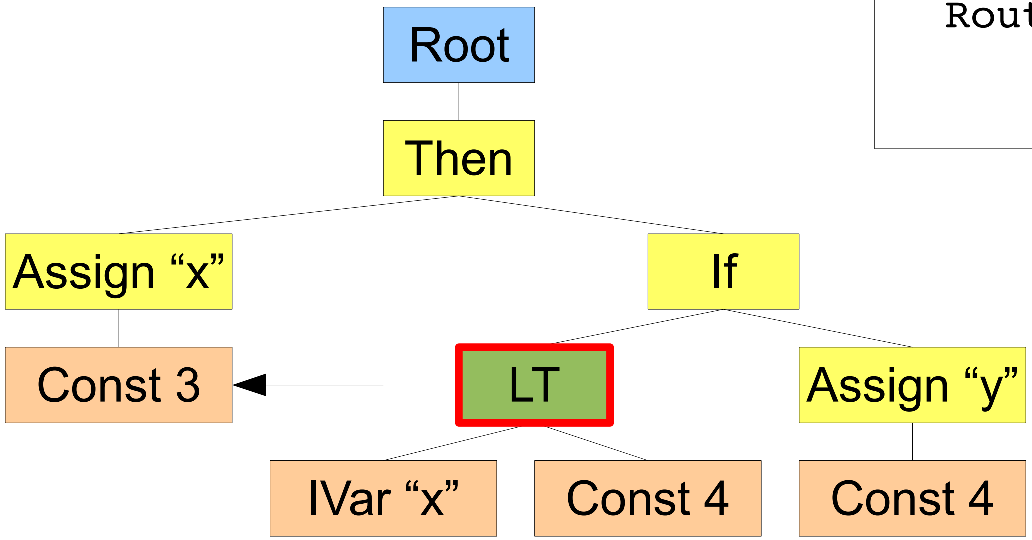
Bookmarks




```
data Route from to where
  Route ::  Path (MovementI Up) from mid →
            Path (MovementI Down) mid to →
            Route from to
```

```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

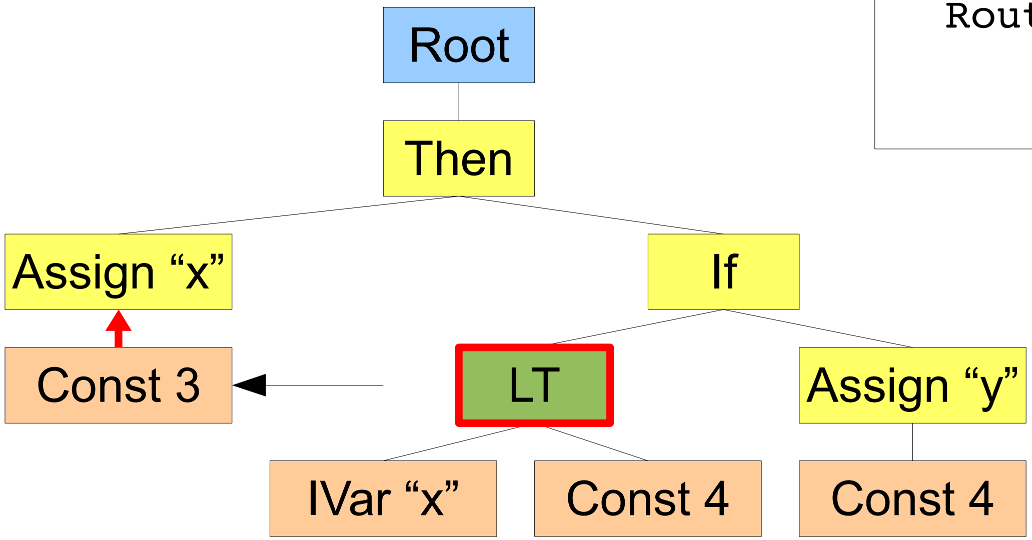
Bookmarks




```
data Route from to where
  Route ::  Path (MovementI Up) from mid →
            Path (MovementI Down) mid to →
            Route from to
```

```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

Bookmarks



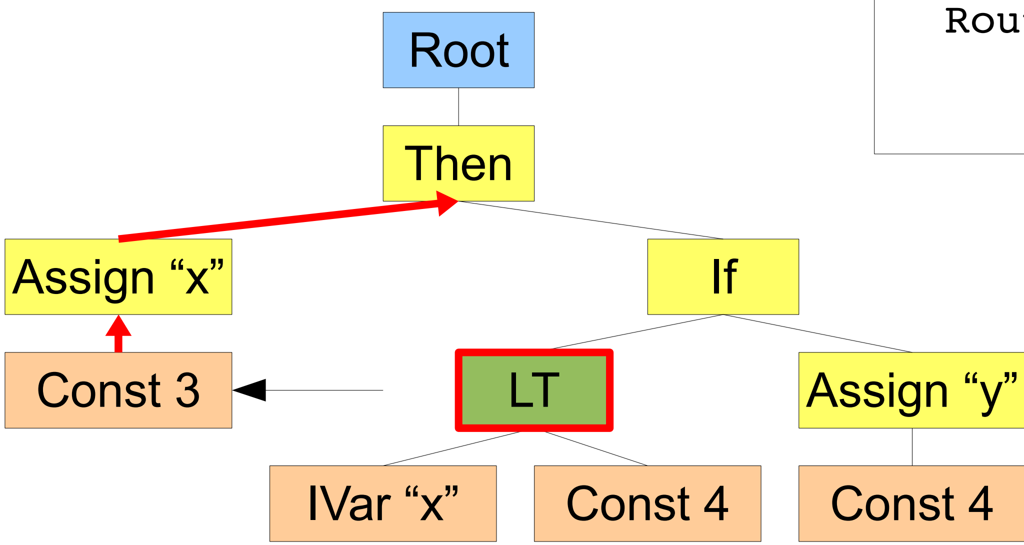

```
data Route from to where
  Route ::  Path (MovementI Up) from mid →
            Path (MovementI Down) mid to →
            Route from to
```

```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

`MAssignToIExp :: MovementI Down Exp IExp`	
`MUp :: MovementI Down b a → MovementI Up a b`	

```
Route (Step (MUp MAssignToIExp) Stop)
      (Stop) :: Route IExp Exp
```

Bookmarks



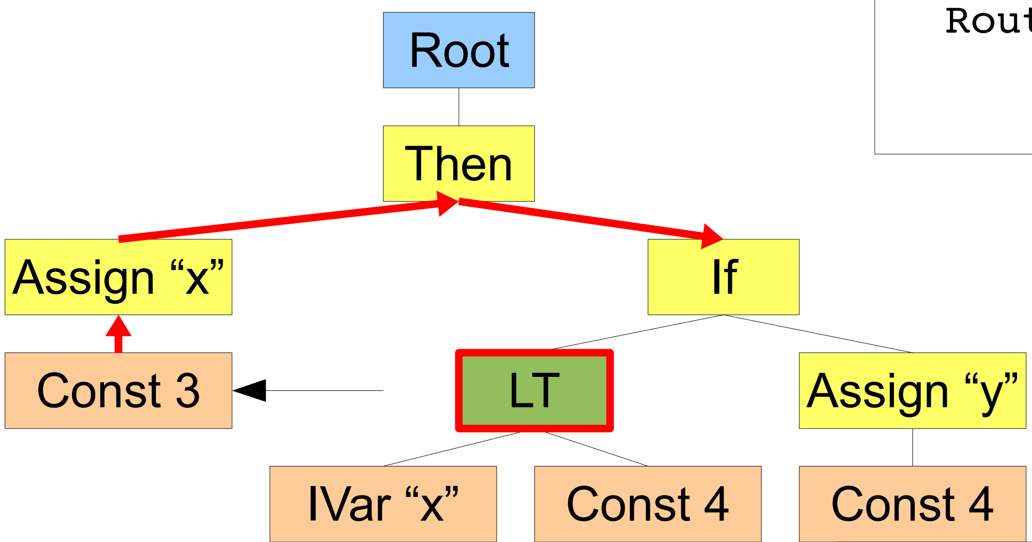

```
data Route from to where
  Route ::  Path (MovementI Up) from mid →
            Path (MovementI Down) mid to →
            Route from to
```

```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

```
MThenToIExp[1] :: MovementI Down Exp Exp
MAssignToIExp :: MovementI Down Exp IExp
MUp :: MovementI Down b a → MovementI Up a b
```

```
Route (Step (MUp MAssignToIExp) (Step (MUp MThenToExp[1]) Stop))
      (Stop) :: Route IExp Exp
```

Bookmarks



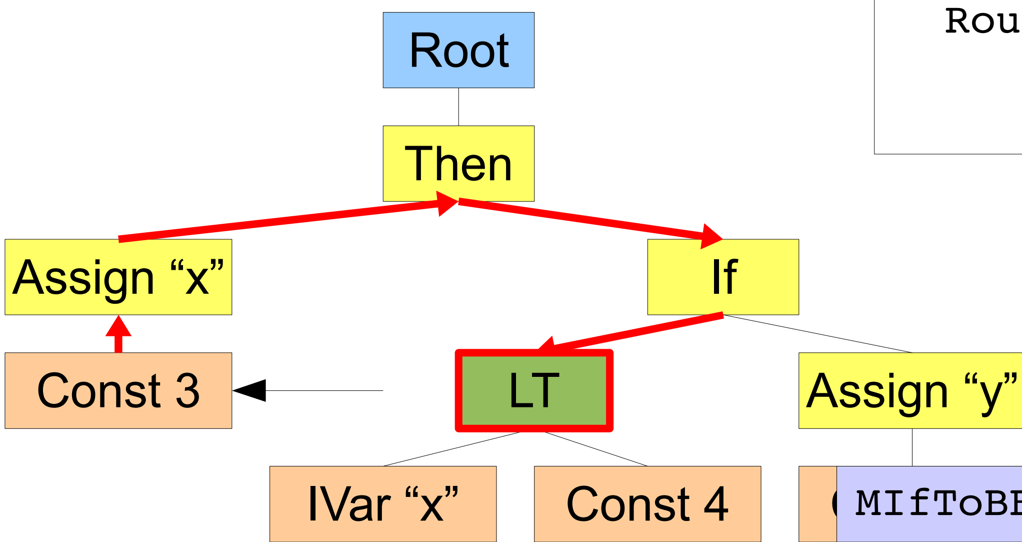

```
data Route from to where
  Route ::  Path (MovementI Up) from mid →
            Path (MovementI Down) mid to →
            Route from to
```

```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

```
MThenToIExp² :: MovementI Down Exp Exp
MThenToIExp¹ :: MovementI Down Exp Exp
MAssignToIExp :: MovementI Down Exp IExp
MUp :: MovementI Down b a → MovementI Up a b
```

```
Route (Step (MUp MAssignToIExp) (Step (MUp MThenToExp¹) Stop))
      (Step MThenToIExp² Stop) :: Route IExp Exp
```

Bookmarks



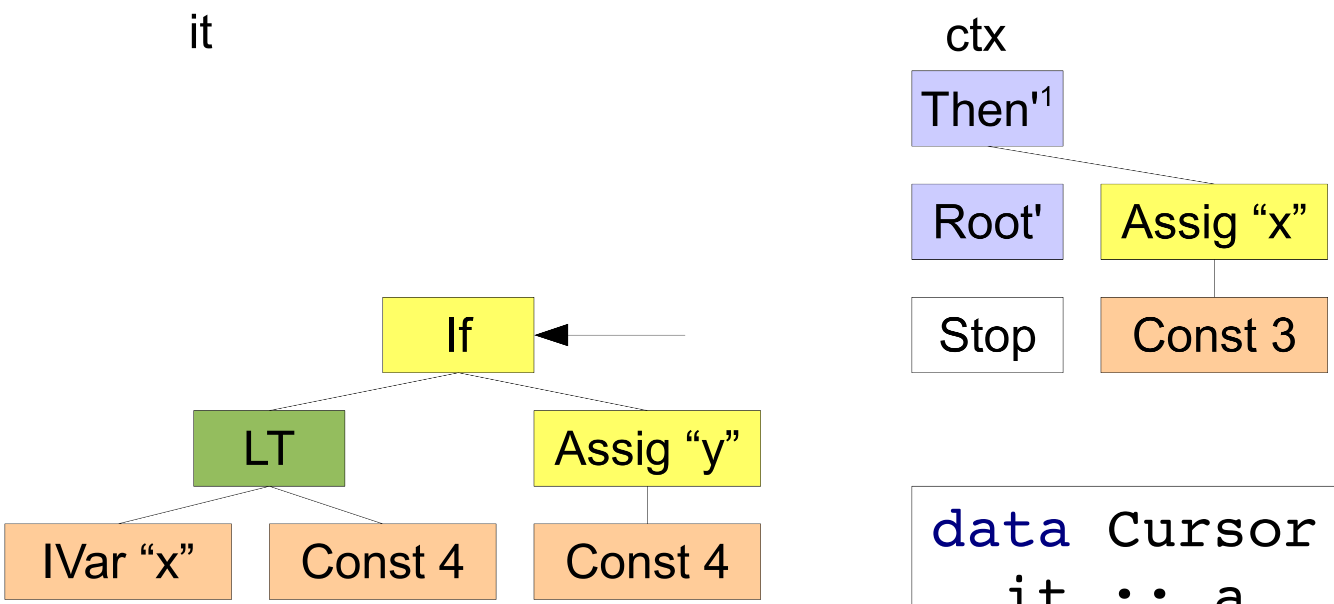

```
data Route from to where
  Route ::  Path (MovementI Up) from mid →
            Path (MovementI Down) mid to →
            Route from to
```

```
data Path tc start end where
  Stop :: Path here here
  Step :: tc start mid →
          Path tc mid end →
          Path tc start end
```

```
MIfToBExp :: MovementI Down Exp BExp
MThenToIExp² :: MovementI Down Exp Exp
MThenToIExp¹ :: MovementI Down Exp Exp
MAssignToIExp :: MovementI Down Exp IExp
MUp :: MovementI Down b a → MovementI Up a b
```

```
Route (Step (MUp MAssignToIExp) (Step (MUp MThenToExp¹) Stop))
      (Step MThenToIExp² (Step MIfToBExp Stop)) :: Route IExp BExp
```

Cursors with Bookmarks

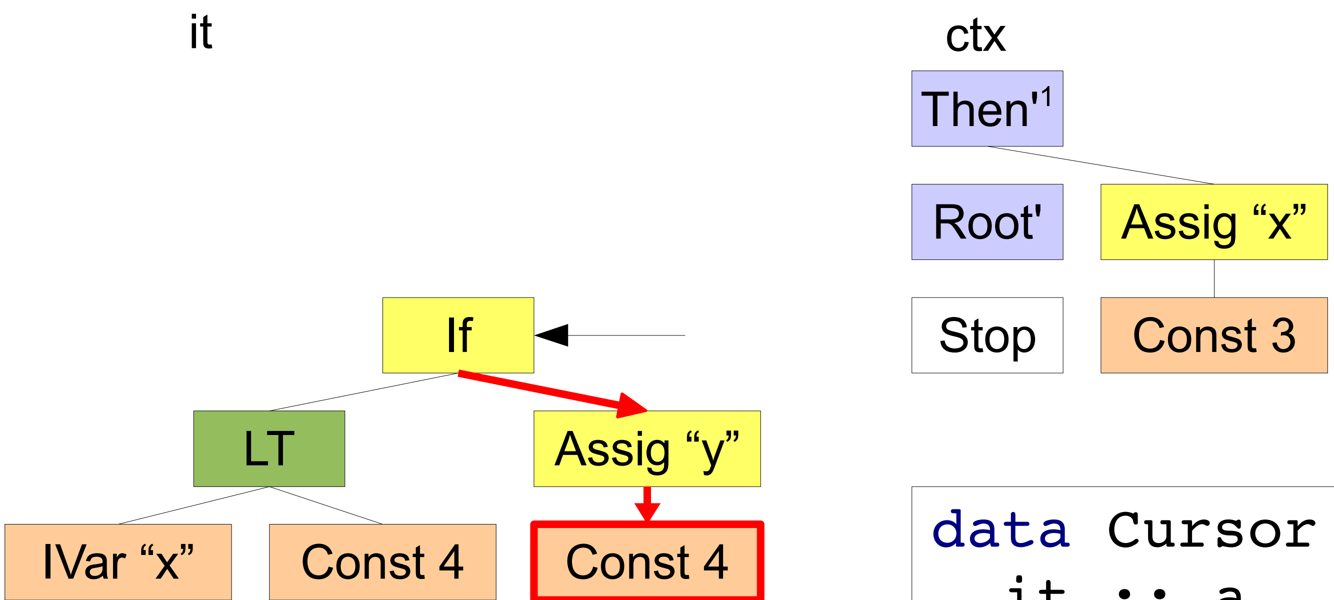



```
data Cursor a = Cursor {
  it :: a,
  ctx :: Path ContextI a Root
}
```

```
Cursor {
  it =  If (LT (IVar “x”) (Const 4)) (Assig “y” (Const 4)),
  ctx = Step (Then'1 (Assig “x” (Const 3))) (Step Root' Stop)

}  :: Cursor Stat
```

Cursors with Bookmarks




```
data Cursor x a = Cursor {
  it :: a,
  ctx :: Path ContextI a Root,
  log :: Route a x
}
```

```
Cursor {
  it =  If (LT (IVar “x”) (Const 4)) (Assig “y” (Const 4)),
  ctx = Step (Then'1 (Assig “x” (Const 3))) (Step Root' Stop),
  log = Route Stop (Step MIfToExp (Step MAssigToIExp Stop))
}  :: Cursor IExp Stat
```

Moving

```
data Cursor x a = Cursor {
    it :: a,
    ctx :: Path ContextI a Root,
    log :: Route a x
  }
```

```
genericMoveUp, genericMoveDown,
genericMoveLeft, genericMoveRight
   :: Cursor x a → Maybe (∃a' . Cursor x a')
```

Writing a GUI

```
data CursorHolder where
  CH :: Cursor a a →
        Map Int (∃a' . Route a a') →
        CursorHolder
```

```
mainLoop cursorHolder = do
    key ← getKey
    let cursorHolder' = onKeyPress key cursorHolder
    render cursorHolder'
    mainLoop cursorHolder'
```

onKeyPress – new bookmark

```
data CursorHolder where
  CH :: Cursor a a →
        Map Int (∃a' . Route a a') →
        CursorHolder
```

```
onKeyPress (KeySaveBookmark i) (CH cursor bookmarks)
   =  let bookmarks' = Map.insert i (Exists emptyRoute)
                                    bookmarks
      in  CH cursor bookmarks'
```

```
emptyRoute :: Route a a
emptyRoute = Route Stop Stop
```

onKeyPress - movement

```
data CursorHolder where
  CH :: Cursor a a →
        Map Int (∃a' . Route a a') →
        CursorHolder
```

```
onKeyPress KeyDown ch@(CH cursor bookmarks)
 = fromMaybe ch $ do
  Exists cursor' ← genericMoveDown cursor
  ...
```

onKeyPress - movement

```
data CursorHolder where
  CH :: Cursor a a →
        Map Int (∃a' . Route a a') →
        CursorHolder
```

```
onKeyPress KeyDown ch@(CH cursor bookmarks)
 = fromMaybe ch $ do
  Exists cursor' ← genericMoveDown cursor
  ...
```

```
∃ o .
  cursor :: Cursor o o     bookmarks :: Map Int (∃a' . Route o a')

  ∃ n .
    cursor' :: Cursor o n
```

onKeyPress - movement

```
data CursorHolder where
  CH :: Cursor a a →
        Map Int (∃a' . Route a a') →
        CursorHolder
```

```
onKeyPress KeyDown ch@(CH cursor bookmarks)
 = fromMaybe ch $ do
  Exists cursor' ← genericMoveDown cursor
  return (CH cursor' bookmarks)
```

```
∃ o .
  cursor :: Cursor o o     bookmarks :: Map Int (∃a' . Route o a')

  ∃ n .
    cursor' :: Cursor o n
```

onKeyPress - movement

```
data CursorHolder where
  CH :: Cursor a a →
        Map Int (∃a' . Route a a') →
        CursorHolder
```

```
appendRoute :: Route a b →
               Route b c →
               Route a c
```

```
onKeyPress KeyDown ch@(CH cursor bookmarks)
 = fromMaybe ch $ do
   Exists cursor' ← genericMoveDown cursor
   let bookmarks' =
     Map.map (\∃ bm → ∃ (log cursor' `appendRoute` bm))
             bookmarks
```

```
∃ o .
  cursor :: Cursor o o    bookmarks :: Map Int (∃a'.Route o a')

  ∃ n .
    cursor' :: Cursor o n  bookmarks' :: Map Int (∃a'.Route n a')
```

onKeyPress - movement

```
data CursorHolder where
  CH :: Cursor a a →
        Map Int (∃a' . Route a a') →
        CursorHolder
```

```
appendRoute :: Route a b →
               Route b c →
               Route a c
```

```
onKeyPress KeyDown ch@(CH cursor bookmarks)
 = fromMaybe ch $ do
  Exists cursor' ← genericMoveDown cursor
  let bookmarks' =
    Map.map (\∃ bm → ∃ (log cursor' `appendRoute` bm))
            bookmarks
  return (CH cursor' bookmarks')
```

```
∃ o .
  cursor :: Cursor o o     bookmarks :: Map Int (∃a'.Route o a')

  ∃ n .
    cursor' :: Cursor o n  bookmarks' :: Map Int (∃a'.Route n a')
```

onKeyPress - movement

```
data CursorHolder where
  CH :: Cursor a a →
        Map Int (∃a' . Route a a') →
        CursorHolder
```

```
resetLog :: Cursor x a →
            Cursor a a
```

```
onKeyPress KeyDown ch@(CH cursor bookmarks)
 = fromMaybe ch $ do
  Exists cursor' ← genericMoveDown cursor
  let bookmarks' =
    Map.map (\∃ bm → ∃ (log cursor' `appendRoute` bm))
            bookmarks
  return (CH (resetLog cursor') bookmarks')
```

```
∃ o .
  cursor :: Cursor o o     bookmarks :: Map Int (∃a'.Route o a')

  ∃ n .
    cursor' :: Cursor o n  bookmarks' :: Map Int (∃a'.Route n a')
```

onKeyPress - movement

```
data CursorHolder where
  CH :: Cursor a a →
```

```
resetLog :: Cursor x a →
```

So the Cursor design means the invariant:

Bookmarks stay in sync with focus

Can be partially encoded in the type system

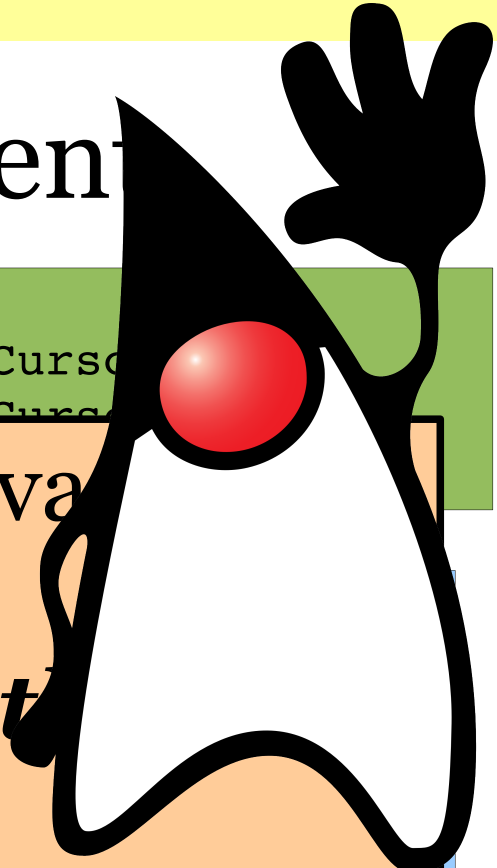
and checked statically by the compiler

```
∃ n .
  cursor' :: Cursor o n  bookmarks' :: Map Int (∃a'.Route n a')
```

onKeyPress - movement

```
data CursorHolder where
  CH :: Cursor a a →
        Map Int (∃a' . Route a a') →
```

```
resetLog :: Curso
            Curso
```



So the Cursor design means the inva

Bookmarks stay in sync wit

Can be partially encoded in the type system

and checked statically by the compiler

```
∃ n .
   cursor' :: Cursor o n  bookmarks' :: Map Int (∃a'.Route n a')
```

and checked statically by the compiler

```
∃ n .
   cursor' :: Cursor o n  bookmarks' :: Map Int (∃a'.Route n a')
```

onKeyPress - select bookmark

```
onKeyPress (KeyLoadBookmark i) ch@(CH cursor bookmarks)
 = fromMaybe ch $ do
  Exists route ← Map.lookup i bookmarks
  Exists cursor' ← cursor `followRoute` route
  let bookmarks' =
    Map.map (\∃ bm → ∃ (log cursor' `appendRoute` bm))
            bookmarks
  return (CH (resetLog cursor') bookmarks')
```

```
followRoute :: Cursor x a → Route a c → Maybe (Cursor x c)
```

CLASE also supports...

- Automatic generation of Context and primitive Movement data types from simple data type declarations
- Automatic generation of a closed reflection scheme for user data types
- Adapters to help with traversals of a Cursor, suitable for (e.g) rendering a Cursor
- Interface to add bound information to traversals
- Simple Persistence (Read/Show) for Cursors

Thank you for listening!

www.zonetora.co.uk/clase/